Property Preserving Encryption in NoSQL Wide Column Stores

Tim Waage and Lena Wiese

Georg August Universität, Göttingen, Germany,
{tim.waage|lena.wiese}@informatik.uni-goettingen.de
Georg August Universität Göttingen, Institute of Computer Science,
Research Group Knowledge Engineering,
Goldschmidtstraße 7, 37077 Göttingen, Germany

Abstract. Property preserving encryption (PPE) can enable database systems to process queries over encrypted data. While a lot of research in this area focusses on doing so with SQL databases, NoSQL (Not only SQL) cloud databases are good candidates either. On the one hand, they usually provide enough space to store the typically larger ciphertexts and special indexes of PPE-schemes. On the other hand in contrast to approaches for SQL systems, despite PPE the query expressiveness remains almost unaffected. Thus, in this paper we investigate (i) how PPE can be used in the popular NoSQL sub-category of so-called wide column stores in order to protect sensitive data in the threat model of a persistent honest-but-curious database provider, (ii) what PPE schemes are suited for this task and (iii) what performance levels can be expected.

Keywords: database security, NoSQL databases, property preserving encryption, wide column stores

1 Introduction

In times of the "Web 2.0" [1] traditional SQL-based database services struggle with the changing demands arising in distributed cloud environments. They are not well suited to represent loosely structured data items like they are typical for the Web 2.0. As NoSQL databases [2, 3] were designed for meeting those new requirements, they attracted more and more attention in the last years, especially in the sub-category of so-called wide column stores (WCS, see Section 3.1). Popular companies developed their own solutions, e.g. Google developed "Bigtable" [4] (used in over 60 Google services) and Facebook developed "Cassandra" [5]. Nowadays it is common to use WCSs on a smaller scale, too. Many cloud database providers offer flexible on-demand services with simple web interfaces for running WCSs remotely in their clusters (e.g. the Google Cloud Platform, Microsoft Azure or Amazon EC2) to exploit the well known benefits of outsourcing databases.

However, storing and processing sensitive data on infrastructures provided by a third party increases the risk of unauthorized disclosure if the infrastructure is compromised by an adversary. Unfortunately, WCSs usually lack security

features like access control, which an external front end or a firewall is assumed to take care of. Hence there is a strong need for providing security and privacy guarantees, because there are several ways how sensitive data can be leaked. An adversary can exploit software vulnerabilities, curious or malicious administrators at hosting providers can snoop on private data or attackers with physical access to servers can steal data from disk and memory [6]. Various examples show, that these threats are not only theoretical [7–10].

The straight-forward solution to reduce the damage caused by server compromises is encrypting the data on a trusted client before it gets uploaded to the cloud servers, then process queries by reading it back from the server to the client, decrypt it, and process the query on the client machine. Unfortunately this requires transferring much more data than necessary (typically large fractions of data are read in order to create relatively small data aggregations) and moves a major part of query computation to the client, which defeats the general purpose of (remote) databases.

Existing approaches (see Section 2) make use of property-preserving encryption (PPE, see Section 3.2) to enable query execution over encrypted data, but the vast majority of existing solutions focusses only on SQL-based technologies and avoids PPE schemes, that rely on index data structures to improve their performance. Hence, this paper makes the following contributions:

- It identifies the requirements for utilizing PPE in the context of WCSs.
- It surveys the practical feasibility of various schemes for order-preserving and searchable encryption with focus on these requirements.
- It shows how PPE can be applied to the data model of WCSs in the honest-but-curious adversary scenario, meaning the database server carries out its tasks as expected, but tries to learn about the data it hosts.
- It implements selected schemes and conducts a practical and comprehensive benchmark using the currently most popular WCSs [11] Cassandra [5] and HBase [12] as underlying platforms. In order to obtain practically relevant results these databases remain unmodified. Hence, we match the conditions that can be found in today's cloud database provider's offers for quantifying the performance loss due to PPE.

2 Related Work

This section surveys related work, limited to approaches that are also designed for the honest-but-curious adversary model, compute over encrypted data and rely on encryption to provide data confidentiality.

Approaches for Relational Databases. One of the first approaches to processing queries over encrypted data is from [13]. Unfortunately in their approach hardware requirements on client side were similar to the ones on the server side. The most popular work on performing queries over encrypted data is "CryptDB" [14] for MySQL and PostgreSQL. It was the first system that could be considered practical, introducing a variety of innovative features, most importantly:

the onion layer model (OLM), which this work uses as well in an improved adaptation (see Section 5.1). However, it uses only PPE schemes that are slow for querying, because the authors avoid (client or server side) indexes. Thus, CryptDB does not scale well when datasets reach a certain size. However, it still receives a lot of scientific attention, in favourable [15, 16] as well as critic ways [17]. "Monomi" [18] can be considered being an extension of CryptDB, trying to support arbitrary SQL queries with the cost of higher requirements for the client machine. "BlindSeer" [19] addresses efficient sub-linear searches for SQL-queries that can be represented as a monotone boolean formula, consisting of the search conditions: keyword match, range and negation. "DBMask" [20] enforces access control cryptographically, based on attribute based access control and combining broadcast and hierarchical key management. It also uses PPE, but not in an OLM-fashion. The authors of "L-EncDB" [21] propose to use so-called format-preserving encryption to realize fuzzy searches.

Approaches for Non-relational Databases. An approach for executing queries over encrypted triple patterns using SPARQL is presented by [22]. Unfortunately, the number of cryptographic keys in their approach is high and every plaintext triple results in eight ciphertext triples, which leads to much overhead in terms of processing and storage inefficiency. To build a distributed key-value store the recent approach of [23] introduces an additional architectural component called dispatcher, that distributes encrypted data to all the database server nodes evenly. Another very recent approach is "Arx" [24] on top of MongoDB, which introduces two proxy servers and needs to know in advance what operations are to be performed on what fields in order to maintain the required indexes.

Hardware Architectures for Encrypted Databases. "Cipherbase" [25] is an extension of Microsoft's SQL Server with two modified parts: the ODBC driver at the client side and the query processor at the server side, that integrates a secure coprocessor within a so-called "trusted machine", realized utilizing field programmable gate arrays (FPGAs). "TrustedDB" [26] is a similar approach, but with tamper-proof cryptographic coprocessors (SCPUs) instead of FPGAs. Even though hardware approaches like these overcome some limitations of CryptDB-based techniques (in particular regarding the query expressiveness), they rely on expensive trusted hardware and require the database to have the user's decryption keys.

3 Background

3.1 The Data Model of Wide Column Stores

WCSs are inspired by Googles BigTable architecture [4], but there are also publicly available open source databases, that rely on the same or a very similar data model, e.g. Hypertable [27], Accumulo [28] as well as used for practical experiments in this work: Cassandra [5] and HBase [12].

WCSs use tables, rows and columns like traditional relational (SQL-based) databases. However, the fundamental difference is that columns are created for

each row instead of being predefined by the table structure. Every row has at least one mandatory column containing its identifier[1]. Except for this column, two rows of the same table can have completely disjunct sets of columns. The identifier of a row has to be unique for the whole table and cannot be used by another row.

Rows are maintained in lexicographic order by their identifier. As WCSs are distributed systems, ranges of such row identifiers serve as units of distribution. Hence, similar row identifiers (and thus data items that are likely to be semantically related to each other) are always kept physically close together, so that reads of ranges require communication to a minimum number of machines. Because row identifiers are used for coordinating distribution this way, changing them would result in changing the row's physical position within the database (cluster), which is prohibitively expensive and thus not allowed. Thus, a row identifier cannot be changed after the row was inserted.

3.2 Property-preserving Encryption

The types of PPE relevant for this work are deterministic encryption (DET), order-preserving encryption (OPE) and searchable encryption (SE). Other types of PPE would have no value for supporting encrypted queries in WCSs. In particular, homomorphic encryption is not considered. Apart from its runtime deficits [18, 29], query mechanisms of WCS would benefit either only minimal (e.g. only SUM() and AVG() in Cassandra's query language) or not at all (e.g. HBase).

DET. The purpose of DET is enabling the database server to check for equality by mapping identical plaintexts to identical ciphertexts.

OPE. The purpose of OPE is enabling a server to learn about the relative order of data elements without gaining information about their exact values. Therefore it encrypts two elements p_1, p_2 of a domain D in a way that $p_1 \leq p_2 \Rightarrow Enc(p_1) \leq Enc(p_2)$ for all $p \in D$. Thus, its use cases are sorting and range queries over encrypted data. A lot of OPE schemes have been proposed with different strategies to map a plaintext to a ciphertext domain (e.g. [30–36]).

SE. The purpose of SE is enabling a server to search over encrypted data without gaining information about the plaintext data. Most SE schemes use indexes (e.g. [37–44]), which are encrypted in a way, that only a trapdoor allows for comparing the searchword with the ciphertext. However, there are also schemes, that avoid having an index by embedding the trapdoor in a special format into the ciphertext itself (e.g. [45]).

4 Selecting Practical PPE Schemes

Having introduced the key characteristics of the WCS data model and the tasks of the different kinds of PPE, we now move on to identify actually feasible PPE schemes for our architecture presented in Section 5.

[1] Commonly referred to as "row key". However we use "row identifier" to avoid confusions with cryptographic keys.

4.1 Deterministic Encryption

In contrast to OPE and SE (see below), the only relevant criterion for practicability of DET is the determinism itself. There are plenty of well known DET schemes, that have proven to work well in practice. For this work we use the Advanced Encryption Standard (AES, [46]).

4.2 Order-preserving Encryption

The criteria for OPE are more complex, due to the working principles of OPE and the WCS data model as described in Section 3.1. We focus on five aspects:

I. Ciphertext (im-)mutability. Ciphertexts produced by an OPE scheme are either *mutable*, meaning they may change as more and more input gets encrypted (e.g. in [35], causing re-writes to the database), or *immutable*, meaning they are final (e.g. [34]). encrypting row identifier columns of WCS tables with OPE requires immutable ciphertexts, as discussed in Section 3.1. However, other columns can be encrypted using mutable schemes.

II. Need for additional data structures. If they are not stateless, OPE schemes require additional data structures for storing their state. That can be done using indexes, trees, dictionaries etc., either on client side (or at least a trusted environment), e.g. [35], or on server side, e.g. [33, 47]. In particular maintaining tree structures is expensive for WCSs.

III. Need for additional architectural components. Some OPE schemes require components running co-located to the database server (e.g. [33]), which cannot be considered practical due to the architectural overhead, that is usually not covered by today's cloud database providers.

IV Input capabilities. Some OPE schemes require detailed knowledge of all plaintexts before encryption (e.g. [32]), which is hard to realize in practical scenarios as databases may grow unpredictably over time. Some schemes even need to encrypt the whole plaintext space in advance [34, 48], instead of encrypting only the desired values on demand.

V Security. Nearly every OPE scheme comes with its own or no formal security definition (see Table 1). Practical security issues resulting from the database scenario and the efforts for corresponding counter measures (e.g. plaintext shifting [31] or using fake queries to hide the data distribution [49]) have to be taken into account too.

Table 1 shows an overview and brief evaluation of the schemes that we have investigated using the above described criteria. Based on this, we selected to implement the schemes of [31] (modular OPE = *"mOPE"*), [34] (Random Subrange Selection = *"RSS"*) and [35] (Optimal Average-Complexity Ideal-Security = *"OACIS"*) for further experiments (see Section 6.2).

To give an idea of why other OPE schemes from Table 1 have been considered impractical, we point out a few of their characteristics that cannot be read from this table: The approaches of [51] and [48] require splitting and partitioning of the plaintext space, causing much metadata overhead. The scheme of [32] requires detailed knowledge of the plaintext space e.g. the smallest distance between

Table 1. Evaluation of the practical feasibility of popular OPE schemes based on the criteria introduced in section 4.2, ordered by date of publication

OPE Scheme	I	II	III	IV	V[1]
[30]	+	−−	+	−	−− (?[2])
[31] (mOPE)	+	++	+	−	+ (POPF-CCA)
[32]	+	−−	+	−−	−− (?[3])
[33]	−	−−	−	++	+ (IND-OCPA)
[34] (RSS)	+	−	+	+	++ (> IND-OCPA[4])
[48]	+	−	+	−	−− (?[2])
[35] (OACIS)	−	−	+	++	+ (IND-OCPA)
[36]	+	+	+	−	++ (> POPF-CCA[4])

[1] IND-OCPA = indistinguishability under ordered chosen-plaintext attack, POPF-CCA = pseudo random order-preserving function against chosen-ciphertext attack (for both, see [50])
[2] only informal security analysis provided by the authors
[3] no security analysis provided by the authors
[4] ">" = proved by the authors to be better than...

two plaintext values, a requirement that hardly can be met in unpredictably growing datasets. As mentioned before the approach of [33] needs an additional component running co-located to the database server, which often is not possible or does not fit to the offers of cloud database providers and causes network communication overhead.

4.3 Searchable Encryption

An evaluation of practical feasibility of the schemes for SE can be done similarly based on the following criteria:

I Need for additional data structures. As mentioned in Section 3.2 SE schemes sometimes use indexes to speed up the search process. These indexes come with the cost of additional pre-processing steps (e.g. selecting keywords, set up the index data structure, etc.), require index maintenance and often an extra round of communication (first for querying the index and then for getting the actual results). Sometimes the underlying index data structure is hardly manageable for WCS databases without much effort (e.g. tree structures).

II Support for Updates. When used in databases, a scheme for performing SE needs the ability to process updates[2], since in most practical cases datasets tend to grow or change. As it turned out, only a surprisingly small number of SE schemes is capable of handling this.

III Algorithm Requirements. Encryption and checking for searchword matches does not work with native database operations like in DET or OPE. Instead,

[2] Mainly meaning adding items to the dataset. SE schemes capable of doing so are commonly referred to as being "dynamic".

more complex procedures are necessary, involving e.g. lookups in auxiliary data structures like bloom filters or traversing trees, using cryptographic primitives or concatenation of strings. For efficiency reasons this should not become too complex.

IV Security vs. V Search Efficiency. Security for index-based (hence, search efficient) schemes and not index-based schemes is hardly comparable due to the information leakage connected to querying the indexes. It consists of information about the index itself (e.g. number of words per document, number of documents, lengths of documents, document-IDs), search pattern information (what was searched for?) and access pattern information (how much answers do I get from executing a certain query compared to executing another one?). This leads to two competitive requirements: avoiding an index leads to more security but is generally slow for searches. Using an index may slow down encryption and leaks additional information, but can speed up querying significantly.

Table 2. Evaluation of the practical feasibility of popular SE schemes based on the criteria introduced in section 4.3, ordered by date of publication

SE Scheme	I	II	III[1]	IV[2]	V[3]
[45] (SWP)	++	++	− (XOR, PRF)	+ (IND-CPA)	$O(n)$
[39]	−−	−−	−− (XOR, SC, DED)	+ (IND-CKA2)	$O(m)$
[40]	−	+	− (XOR)	+ (IND-CKA2)	$O(log(u))$
[41]	−−	++	−− (XOR, PRF, HSH)	+ (IND-CKA2)	$O(log(u))$
[42]	−−	+	−− (XOR, PRF, HOM)	+ (IND-CKA2)	$O(m)$
[43] (SUISE)	−	++	− (SC, PRF)	+ (IND-CKA2)	$O(n/u)$
[44]	−−	−−	−− (XOR, SC, DED)	+ (IND-CKA2)	$O(m)$

[1] XOR = bitwise exclusive OR operations, PRF = pseudo-random functions, SC = string concatenation, DED = deterministic encryption/decryption, HOM = homomorphic encryption, HSH = hash functions

[2] IND-CPA = indistinguishability under chosen-plaintext attacks, IND-CKA2 = adaptive indistinguishability under chosen keyword attacks (for both, see [52])

[3] searching on a dataset with size of n words (u of which are unique), resulting in m matches

Table 2 gives an overview and brief evaluation of the SE schemes that we investigated in regard to the given criteria. Based on it, we selected to implement the schemes of [45] (*"SWP"*, an abbreviation of the three author's names Song, Wagner and Perrig) and [43] (securely updating index-based searchable encryption = *"SUISE"*) for further experiments (see Section 6.2).

Like we previously did for OPE, we point out reasons for why we do not consider other schemes from Table 2 being practical. [37] proposes to use bloom filters as indexes per document. Bit-wise bloomfilter operations are hard to handle for a WCS and a bloomfilter limits the number of words per document. [38] presented a similar approach, but using pre-built dictionaries, which they propose to store either on clientside (which defeats the purpose of remote searchable

encryption) or on server side (which basically has the same shortcomings as in [37]). The approach of [39] achieves sub-linear search time, using a complex and for a database barely manageable index structure. Furthermore, [37–39] do not support updates. Hence, [42] proposed an extension for [39] with two additional laborious serverside data structures. [40] proposes a scheme in which not only searches require multiple rounds of communication between client and server, but also storage. Realizing this problem, the authors of [41] got rid of the interactivity by introducing even more client side computation.

5 Data Management on Server Side

We now have selected feasible PPE schemes, but just storing the PPE-encrypted values would of course leak information instantly (like equality and relative order between values) that is not supposed to leak when not querying. Therefore the authors of [53] proposed a so-called onion layer model (OLM) for their SQL-based architecture "CryptDB". The idea is to encrypt every value with a PPE scheme of each category (DET, OPE and SE) separately that leaks just enough information to still be able to perform certain operations over the encrypted data (as described in Section 3.2) and wrap it into another layer of random encryption (in the following: "RND") for not leaking any information. Then, the RND layer is only removed, if a query requires it. In this way the database is still able to process queries, but it learns only the minimal necessary amount of information.

5.1 An Onion Layer Model for WCSs

We adapt the basic idea of CryptDB's OLM, but the data model of WCSs requires some changes. As explained in Section 3.1, a fundamental working principle of WCSs is keeping all rows of a table sorted by the content of the row identifier column. Thus, the database must be able to compare the row identifier of a new row to be inserted with already existing ones in the table. Therefore OPE columns of row identifiers are not allowed to have a RND layer. Otherwise the WCS data model would be broken. That means, row identifier columns must be treated differently from all other columns regarding the onion layer design. They must leak the order of values to allow for row sorting independent of querying.

All data types supported by the databases can be grouped in three categories: strings (e.g. text, characters), numerical values (e.g. integers, timestamps) or byte blobs (e.g. byte arrays, raw files). Figure 1 presents this work's OLM design for string columns, before (left) and after (right) queries involving equality checks and order comparisons were executed. The upper row shows the onions for regular string columns. Note the missing RND layers after a query like Q3 - Q5 (see Section 6.2) was processed and the SE onion not requiring a RND layer at all, because schemes for SE usually provide the same security guarantees as RND layer encryption. The row below shows the onion in case of the string column is a row identifier column. Note that in this case the DET onion

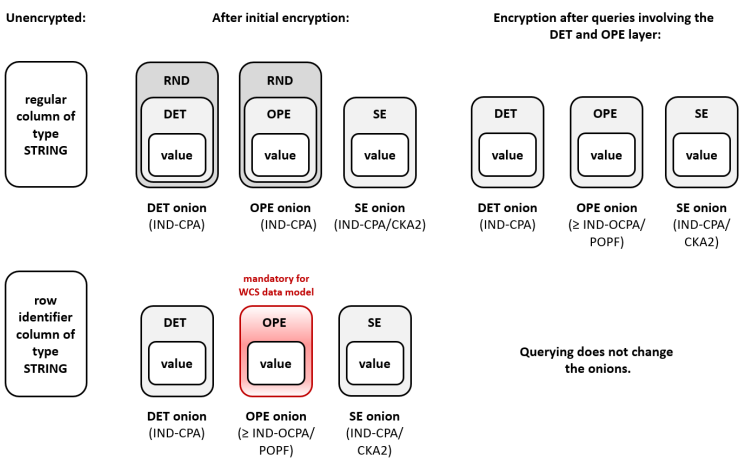

Fig. 1. Transformation of a plain string column into onion-layered ciphertext columns

comes without a RND layer, because the mandatory OPE column already leaks equality[3], rendering a RND layer wrapped around the DET onion useless. However, keeping a DET onion still makes sense, because having implemented AES in hardware on the majority of modern processors, its decryption process works much faster than the decryption algorithms of other PPE schemes.

While Figure 1 illustrates the situation for string columns as most complex cases, the OLM for the other two type categories is different, since not all operations make sense for all types of data. For instance, numerical values do not need the SE onion, because searching for words in numbers is neither somehow defined nor possible, even in unencrypted databases. Thus, numerical values can be encrypted faster than strings in the OLM. We investigate the difference in encryption performance in Section 6.2.

5.2 Querying the Encrypted Data

In our framework, the work flow for executing queries against PPE encrypted data organized in an OLM as described in Section 5.1 is as follows.

Step 1: A query usually contains one or more conditions that have to be met by a row to be included in the result set. Such a condition is always of the form *[columnname, compare operation, comparator]*. For every condition the client checks which onion is involved and whether the RND layer of this onion still exists and has to be removed or not (e.g. if the compare operation is an equality check indicated by an "=", the column representing the DET onion gets checked).

[3] Apart from rare exceptions (e.g. [51]) OPE schemes are deterministic. Hence they leak not only the relative order between plaintexts, but also equality.

Step 2: After all necessary RND layers have been removed by the client (step 1) the set of all columns is identified, that have to read from the database. This set consists of two subsets, that might overlap. The first subset consists of all onion columns that are involved in query conditions as described above. The second subset consists of all columns, that were selected by the user's query.

Step 3: Having collected that information, the client constructs the query against the encrypted database by doing the following. All plaintext column names are replaced with their ciphertext counterparts according to the information that was collected previously (step 2). Furthermore all plaintext literals in conditions are replaced by their PPE encrypted counterparts. The query now contains no plaintext information anymore and can be carried out by the server.

Step 4: The client receives and decrypts the results. Note that the removed RND layers stay off, because the database saw the underlying values anyway.

6 Evaluation

6.1 Implementation

All experiments in this section were run on an Intel Core i7-4600U CPU @ 2.10GHz, 8GB RAM, a Samsung PM851 256GB SSD using Ubuntu 16.04. The PPE schemes were implemented in Java 8, using cryptographic primitives of the Java Cryptography Extension and The Legion of the Bouncy Castle Java Cryptography API[4]. In order to avoid measuring network effects local installations of the databases were used, as only the computation time of the schemes in combination with the speed of the databases was to be measured. Cassandra was used in version 3.9, Apache was used in version 1.3.

6.2 Performance

While AES is a performant option for encryption and decryption in the RND and DET layer, we evaluated the performance as well as strengths and weaknesses in previous work for the OPE [54] and SE [55] layer schemes. Depending on the application the user might want to exploit these strengths and minimize the impact of these weaknesses. That is why we group the schemes that we have selected in Section 4 in three profiles, each of which determines what PPE schemes are actually used in the OPE and SE layer during data insertion:

- `OPTIMIZED READING`: This profile prioritizes schemes that have advantages for read queries (e.g. like selections). Thus, it is the best choice for "write-once" databases. The OPE schemes best suited for fast reading are RSS and OACIS. They have the same type of index, which results in equal reading performance. However, RSS is the preferred choice for this profile, because it has some minor advantages in the encryption process. For the SE layer the SUISE scheme is used. It is faster than SWP in the process of searching, in particular for repeated queries.

[4] available at `https://www.bouncycastle.org/`

- `OPTIMIZED WRITING`: This profile prioritizes schemes with fast encryption algorithms for scenarios, in which data insertion occurs more often than reading. The OPE scheme best suited for fast writing is OACIS. For the SE layer the SWP scheme is used. Since it does not have to maintain indexes, it can insert data faster than SUISE.
- `STORAGE-EFFICIENT`: This profile prioritizes storage needs over computation time and hence PPE schemes, that require the least amount of storage for data and indexes on client and server side. Hence, it uses mOPE for the OPE layer and SWP for the SE layer, both working without indexes.

Not affected by these profiles is only the OPE layer of row identifier columns, that must always be encrypted using RSS. The other schemes are not suited for row identifiers for the following reasons. OACIS cannot be used, because it produces mutable ciphertexts, but row identifiers are supposed to be final (see Section 3.1). mOPE cannot be used, because it adds a secret modular offset to the ciphertexts, that the database must not know about, but that has to be taken into account, when inserting values.

PPE scheme indexes (if existing) are maintained per column. This allows involving only the index data of columns actually required for answering a query.

For our tests we used a subset of the Enron email dataset[5], which reflects the practical scenario of using PPE for protecting sensitive mailbox data. We assume an average mailbox to have a size from 1,000 up to 10,000 emails. Hence, the measurements are started with 1,000 randomly chosen emails of the corpus, which we increase up to 10,000 emails (that contain $1.03 \cdot 10^7$ words in 180.000 fields of data) in order to estimate how the schemes and databases scale.

Encryption. We measure the time for encrypting and inserting up to 10,000 mails ($1.03 \cdot 10^7$ words) using the table profiles and OLM as introduced above and presented the results in Table 3. Since the complexity of encrypting different types of data in the OLM is different, we distinguish between the encryption of text and numerical data. Hence, in a first series of measurements we investigate the performance of text data, using the data fields of the Enron email dataset "as they are", meaning as strings, that are encrypted as shown in Figure 1. Note that this is the worst case scenario for the proposed architecture, because OPE for strings and SE layer computations are the most expensive operations in the OLM. These measurements are indicated in Table 3 as "text data". In a second series of measurements we extract an equal amount of numerical values from the Enron emails, like their size, date, priority etc. Their OLM encryption is less complex, because OPE-encryption can be computed faster (since the necessary numerical format is already given) and the SE layer is not involved at all. [6]

As could be expected, the profile for `OPTIMIZED WRITING` performs best, increasing the average insertion time for text data by a factor of 5.92 using Cassandra and 5.62 using HBase. When inserting numerical data, the performance

[5] available at `https://www.cs.cmu.edu/~./enron/`

[6] We do not perform test for byte blob data, since in the proposed OLM this would only result in performing in AES encryption.

is much less affected by encryption: 35% with Cassandra and even only 10% with HBase. The profile for `OPTIMIZED READING` performs well for numerical data, roughly doubling the insertion times, whereas the performance loss factors are 26.1 and 22.9 for text data. The `STORAGE EFFICIENT`-profile suffers from the slow mOPE scheme in the OPE layer, which results in overall performance loss factors of 40.4/34.5 and 10.4/8.3. In real-world scenarios the performance will be somewhere in between the extremes for text and numerical data, that are shown in Table 3, depending on the composition of the dataset. In summary, it can be said that the `OPTIMIZED WRITING`-profile is a a justifiable option in relation to the security that can be gained. Since in most scenarios query performance is much more crucial than writing performance, even the profile for `OPTIMIZED READING` should be sufficient for the majority of use cases. However, the `STORAGE EFFICIENT`-profile might not be feasible in practice due to its computationally expensive PPE schemes.

Concerning the databases it can be observed, that there are only non-significant differences. HBase is always a little faster than Cassandra. The smallest difference (5.3%) between both occurs when using the profile for `OPTIMIZED WRITING` for text data, the biggest difference (25.3%) when using `STORAGE EFFICIENT`-profile for numerical data.

Table 3. Time needed for onion layer encryption of 180.000 fields of text data and 180.000 fields of numerical data in seconds (OR = Optimized Reading, OW = Optimized Writing, S = Storage-efficient, PL = performance loss).

Database		text data			numerical data		
Cassandra	unencrypted	9.71			8.91		
	profile	OR	OW	S	OR	OW	S
	encrypted	253.2	57.48	393.0	22.2	12.1	93.1
	factor of PL	26.1	5.92	40.4	2.49	1.35	10.4
HBase	unencrypted	10.5			10.1		
	profile	OR	OW	S	OR	OW	S
	encrypted	241.0	59.01	362.2	20.6	11.1	83.8
	factor of PL	22.9	5.62	34.5	2.04	1.1	8.3

Querying. Querying is more complicated than encrypting. The runtime of a query depends on many aspects, for example the query type, the state of the onion layers and the PPE schemes used. These aspects are considered during the benchmarks in the following ways. First of all, five queries (Q1 - Q5) are tested, that involve dealing with different combinations of PPE schemes and are based on real world use cases. Q1 asks for all emails of a certain sender. Thus, it requires one check for equality and involves the DET layer once (see Section 3.2). Q2 asks for all emails larger than a certain size. Hence it requires OPE once, because the order relation between two values has to be determined (also,

see Secction 3.2). Q3 asks for all emails with a certain word in their body. That means a word has to be searched for in encrypted text. That involves the SE layer once (again, see Section 3.2). Q4 combines all filter criteria from Q1 - Q3 and Q5 is a more complex query, that asks for all mails from a certain sender (DET) in a certain period of time (2x OPE for the start and end point of this period) with certain words in the subject and body fields (2x SE). When performing one of them, it makes a significant difference in terms of runtime whether the same or a similar query was performed before or not, which the following two reasons are responsible for. Firstly, columns being involved in an equality check or order comparison before already lost their RND layer. The effort of removing it is not necessary again. Secondly, when the SUISE scheme is involved in SE more than once, it might already have the results in a second index it maintains, which also improves the query runtime significantly. For these two reasons every query is executed twice with Qx.1 indicating the first execution of the query Qx after encryption and Qx.x indicating the runtime, that can be expected from all future executions of Qx.

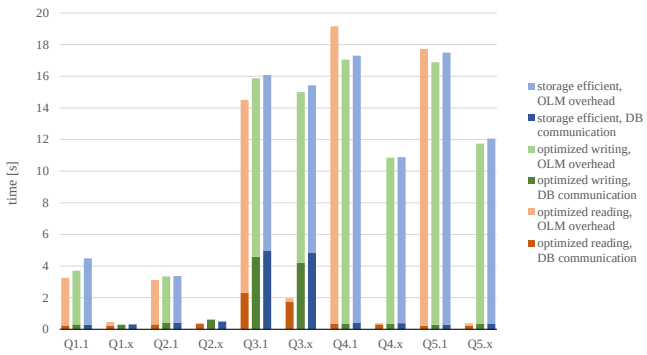

Fig. 2. Query runtime with Cassandra

The results are presented in Figure 2 for Cassandra and Figure 3 for HBase. In these figures "DB communication" denotes the pure runtime of the databases' communication mechanisms (which is the execution time of driver calls in case of Cassandra and the execution time of HBase's native API calls) and "OLM overhead" denotes everything that is a direct or indirect consequence of the onion layer model, for example query rewriting, RND layer removal or the SE processing as described in Section 5. Decryption time of the resultset (as obtained in step 4 of Section 6.2) is not explicitly shown, since it is insignificant (under 5ms for all queries). All queries have been conducted with the largest dataset that has been used in the previous encryption benchmark, having a volume of $1.03 \cdot 10^7$ words in 10,000 emails.

The following observations can be made. If encryption was done using the profile for `OPTIMIZED READING`, all Qx.x queries perform well under one sec-

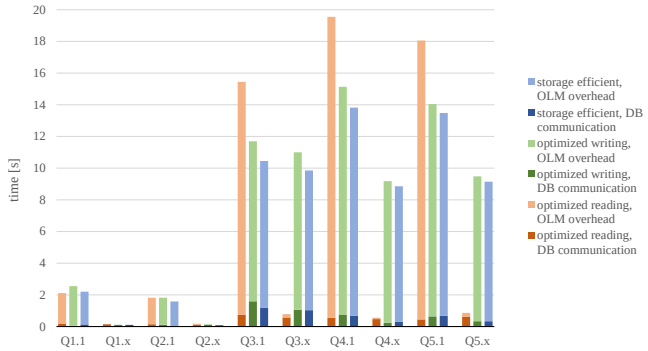

Fig. 3. Query runtime with HBase

ond, except for Q3.x in combination with Cassandra. This can be considered practically feasible performance [15]. Qx.x queries are always faster than Qx.1 queries. That means the performance always improves, if similar queries are executed. Performing SE is very expensive compared to requiring DET or OPE functionality only. However, performing SE on small fields of data has barely a performance impact (compare Q4 and Q5; Q4 searches in bodies, Q5 adds SE only in the small subject field), but can slow down querying significantly, if done on a large subset of the data (compare Q1 and Q2 against Q3). HBase seems to have a slight overall performance advantage. A possible explanation for this might be the RND layer removal, which HBase is doing almost twice as fast compared to Cassandra. This can be seen in Q1 and Q2, in which most of the time is need for the RND layer removal.

7 Security

A formal security analysis of the PPE schemes that we used can be found in their originating publications. In this section we briefly discuss security aspects arising from putting the schemes into practice.

For AES, the only scheme used and needed in the DET layer, there are neither known attacks of practical relevance, nor any leakage besides the intentional determinism.

In SE schemes mainly three kinds of information can leak unintentionally: index information (e.g. number of words per document, number of documents, lengths of documents, document-IDs), search pattern information (what word was searched for?) and access pattern information (how much answers do I get from executing a certain query compared to executing another one?). The leakage of the used SE schemes can be described as follows. SWP [45] does not need an index and thus does not leak any index information and also hides the true length of plaintext words, because it uses fixed length trapdoors. However it leaks search patterns, since it passes pre-encrypted search words to the database,

that are deterministically encrypted, which allows linking them to actual plain words. SWP also leaks access patterns, which is unavoidable in the client server scenario, in which a request (pre-encrypted searchword) can always be linked to the corresponding answer (the result set). The SUISE scheme [43] leaks the number of unique words per document. Since search tokens are generated deterministically, the search pattern leaks like in SWP. Trapdoors have a constant length, which also hides the length of the plaintext words.

Concerning the OPE layer neither of the schemes can leak any index information, because the index (if exists) resides on clientside. Search patterns can leak, because the used OPE schemes produce deterministic ciphertexts that can be tracked. Access patterns leak for the same reasons explained earlier for SE. Another security risk for OPE is to encrypt either very few or very much values of a domain: on the one hand, if only two values of a domain p_1 and p_2 are encrypted, they can easily be mapped to their corresponding ciphertexts c_1 and c_2. Obviously the smaller p value is encrypted in the smaller c value and the larger p value is encrypted in the larger c value. On the other hand, it is equally severe if all values of a specific domain have been encrypted. The ordered ciphertexts can simply be mapped to the ordered plaintexts (note that both problems also occur in non-deterministic OPE schemes). That means e.g. it makes sense to store a date in form of a unix timestamp, not split in individual characteristics like day (domain size only 31), month (domain size 12), etc.

8 Extensions and Future Work

The topic of this work leaves a lot of room for additional work. An evaluation in a real world cloud environment is needed. Fragmentation over independent databases (which we already implemented, but not presented for reasons of complexity) can be of further use to impede statistical attacks based on PPE leakage. We also plan to support other databases and data models. WCS tables can easily be transformed to fit e.g. in key value or document stores. An additional onion for homomorphic encryption can be used for data aggregations like sums and averages. Schemes for fuzzy/similarity search are of interest for the SE layer.

9 Conclusion

We analysed the requirements that PPE schemes have to meet in order to be feasible for NoSQL WCSs and evaluated various available schemes for OPE and SE regarding these requirements. Based on our findings we identified feasible PPE schemes, proposed an OLM to handle PPE encrypted data on serverside and implemented both for a practical evaluation using the two popular WCSs Cassandra and HBase. We quantified the performance impact of PPE encryption and characterized practical security issues. We showed that choosing PPE schemes corresponding to the read/write needs of the scenario leads to still practically feasible performance.

References

1. O'Reilly, Tim: What is web 2.0: Design patterns and business models for the next generation of software. Communications and Strategies **65**(1) (2007) 17–37
2. Han, J., Haihong, E., Le, G., Du, J.: Survey on NoSQL database. In: Pervasive computing and applications (ICPCA), 2011 6th International Conference on, IEEE (2011) 363–366
3. Tudorica, B.G., Bucur, C.: A comparison between several NoSQL databases with comments and notes. In: Roedunet International Conference (RoEduNet), 2011 10th, IEEE (2011) 1–5
4. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems **26**(2) (2008) 4
5. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Systems Review **44**(2) (2010) 35–40
6. Alex, H., Schoen, S., Heninger, N., Clarkson, W., Paul, W., Calandrino, J., Feldman, A., Appelbaum, J., Felten, E.: Lest we forget - cold boot attacks on encryption keys. (2008)
7. Corkery, M.: Once Again, Thieves Enter Swift Financial Network and Steal, New York Times (2016) `http://www.nytimes.com/2016/05/13/business/dealbook/swift-global-bank-network-attack.html`, checked: 20/03/2017.
8. Lennon, M.: Hackers Used Sophisticated SMB Worm Tool to Attack Sony, Security Week (2016) `http://www.securityweek.com/hackers-used-sophisticated-smb-worm-tool-attack-sony`, checked: 20/03/2017.
9. Quinn, B., Arthur, C.: Playstation network hackers access data of 77 million users, The Guardian (2011) `https://www.theguardian.com/technology/2011/apr/26/playstation-network-hackers-data`, checked: 20/03/2017.
10. Crawford, D., Fuhrmans, V., Ball, D.: Germany Tackles Tax Evasion, Wall Street Journal (2010) `http://www.wsj.com/articles/SB10001424052748704197104575051480386248538`, checked: 20/03/2017.
11. Solid-IT: DB-engines ranking (2017) `https://db-engines.com/de/ranking`, checked: 20/03/2017.
12. Borthakur, D., Gray, J., Sarma, J.S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., et al.: Apache hadoop goes realtime at facebook. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, ACM (2011) 1071–1080
13. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of data, ACM (2002) 216–227
14. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: Processing queries on an encrypted database. Communications of the ACM **55**(9) (2012) 103–111
15. Shahzad, F., Iqbal, W., Bokhari, F.S.: On the use of CryptDB for securing electronic health data in the cloud: A performance study. In: 2015 17th International Conference on E-health Networking, Application & Services (HealthCom), IEEE (2015) 120–125
16. Tetali, S.D., Lesani, M., Majumdar, R., Millstein, T.: Mrcrypt: Static analysis for secure cloud computations. ACM SIGPLAN Notices **48**(10) (2013) 271–286
17. Akin, I.H., Sunar, B.: On the difficulty of securing web applications using CryptDB. In: Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on, IEEE (2014) 745–752

18. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In: Proceedings of the VLDB Endowment. Volume 6., VLDB Endowment (2013) 289–300
19. Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A., Bellovin, S.: Blind seer: A scalable private dbms. In: 2014 IEEE Symposium on Security and Privacy, IEEE (2014) 359–374
20. Sarfraz, M.I., Nabeel, M., Cao, J., Bertino, E.: Dbmask: fine-grained access control on encrypted relational databases. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, ACM (2015) 1–11
21. Li, J., Liu, Z., Chen, X., Xhafa, F., Tan, X., Wong, D.S.: L-encdb: A lightweight framework for privacy-preserving data queries in cloud computing. Knowledge-Based Systems **79** (2015) 18–26
22. Kasten, A., Scherp, A., Armknecht, F., Krause, M.: Towards search on encrypted graph data. Proc. of PrivOn (2013)
23. Yuan, X., Wang, X., Wang, C., Qian, C., Lin, J.: Building an encrypted, distributed, and searchable key-value store. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ACM (2016) 547–558
24. Poddar, R., Boelter, T., Popa, R.A.: Arx: A strongly encrypted database system. IACR Cryptology ePrint Archive **2016** (2016) 591
25. Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., Venkatesan, R.: Orthogonal security with cipherbase. In: CIDR. (2013)
26. Bajaj, S., Sion, R.: Trusteddb: A trusted hardware-based database with privacy and data confidentiality. Knowledge and Data Engineering, IEEE Transactions on **26**(3) (2014) 752–765
27. Khetrapal, A., Ganesh, V.: Hbase and hypertable for large scale distributed storage systems. Dept. of Computer Science, Purdue University (2006) 22–28
28. Sawyer, S.M., O'Gwynn, B.D., Tran, A., Yu, T.: Understanding query performance in accumulo. In: High Performance Extreme Computing Conference (HPEC), 2013 IEEE, IEEE (2013) 1–6
29. Cooney, M.: IBM touts encryption innovation,Network World (2009) `http://www.networkworld.com/article/2259168/data-center/ibm-touts-encryption-innovation.html`, checked: 20/03/2017.
30. Kadhem, H., Amagasa, T., Kitagawa, H.: A secure and efficient order preserving encryption scheme for relational databases. In: KMIS. (2010) 25–35
31. Boldyreva, A., Chenette, N., ONeill, A.: Order-preserving encryption revisited: Improved security analysis and alternative solutions. In: Advances in Cryptology–CRYPTO 2011. Springer (2011) 578–595
32. Liu, D., Wang, S.: Programmable order-preserving secure index for encrypted database query. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, IEEE (2012) 502–509
33. Popa, R.A., Li, F.H., Zeldovich, N.: An ideal-security protocol for order-preserving encoding. In: Security and Privacy (SP), 2013 IEEE Symposium on, IEEE (2013) 463–477
34. Wozniak, S., Rossberg, M., Grau, S., Alshawish, A., Schaefer, G.: Beyond the ideal object: towards disclosure-resilient order-preserving encryption schemes. In: Proceedings of the 2013 ACM workshop on Cloud computing security workshop, ACM (2013) 89–100
35. Kerschbaum, F., Schröpfer, A.: Optimal average-complexity ideal-security order-preserving encryption. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM (2014) 275–286

36. Chenette, N., Lewi, K., Weis, S.A., Wu, D.J.: Practical order-revealing encryption with limited leakage. (2015)
37. Goh, E.J., et al.: Secure indexes. IACR Cryptology ePrint Archive (2003) 216
38. Chang, Y.C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Applied Cryptography and Network Security, Springer (2005)
39. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications security, ACM (2006) 79–88
40. Sedghi, S., Van Liesdonk, P., Doumen, J.M., Hartel, P.H., Jonker, W.: Adaptively secure computationally efficient searchable symmetric encryption. (2009)
41. Van Liesdonk, P., Sedghi, S., Doumen, J., Hartel, P., Jonker, W.: Computationally efficient searchable symmetric encryption. In: Workshop on Secure Data Management, Springer (2010) 87–100
42. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, ACM (2012) 965–976
43. Hahn, F., Kerschbaum, F.: Searchable encryption with secure and efficient updates. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM (2014) 310–320
44. Jho, N.S., Chang, K.Y., Hong, D., Seo, C.: Symmetric searchable encryption with efficient range query using multi-layered linked chains. The Journal of Supercomputing (2015) 1–14
45. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on, IEEE (2000) 44–55
46. Anderson, R., Biham, E., Knudsen, L.: Serpent: A proposal for the advanced encryption standard. NIST AES Proposal **174** (1998) 1–23
47. Roche, D., Apon, D., Choi, S.G., Yerukhimov, A.: Pope: Partial order-preserving encoding. Technical report, Cryptology ePrint Arch. 2015/1106 (2015)
48. Liu, Z., Chen, X., Yang, J., Jia, C., You, I.: New order preserving encryption model for outsourced databases in cloud environments. Journal of Network and Computer Applications (2014)
49. Mavroforakis, C., Chenette, N., O'Neill, A., Kollios, G., Canetti, R.: Modular order-preserving encryption, revisited. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM (2015) 763–777
50. Boldyreva, A., Chenette, N., Lee, Y., Oneill, A.: Order-preserving symmetric encryption. In: Advances in Cryptology-Eurocrypt 2009. Springer (2009) 224–241
51. Kadhem, H., Amagasa, T., Kitagawa, H.: Mv-opes: Multivalued-order preserving encryption scheme: A novel scheme for encrypting integer value to many different values. IEICE Transactions on Information and Systems **93**(9) (2010) 2520–2533
52. Bösch, C., Hartel, P., Jonker, W., Peter, A.: A survey of provably secure searchable encryption. ACM Computing Surveys (CSUR) **47**(2) (2015) 18
53. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM (2011) 85–100
54. Waage, T., Homann, D., Wiese, L.: Practical Application of Order-Preserving Encryption in Wide Column Stores. In: Proceedings of the 13th International Joint Conference on e-Business and Telecommunications - SECRYPT. (2016) 352–359
55. Waage, T., Jhajj, R.S., Wiese, L.: Searchable encryption in apache cassandra. In: International Symposium on Foundations and Practice of Security, Springer (2015) 286–293